VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Jan Stejskal

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

HOUGHOVA TRANSFORMACE A JEJÍ VARIANTY
HOUGH TRANSFORM AND ITS VARIANTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jan Stejskal**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Kamil Říha, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Jan Stejskal ***ID:*** 211272
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Houghova transformace a její varianty

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte teoretický princip Houghovy transformace a jejích různých variant, jako je transformace přímky, kružnice a obecných tvarů. Naprogramujte aplikaci pro demonstraci jejích vlastností. V aplikaci demonstrujte také detekci různých tvarů v parametrickém prostoru včetně jeho vizualizace. Doporučený nástroj pro implementaci: knihovny OpenCV a MS Visual C++.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ, R. C.; WOODS, R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002.

[2] BRADSKI, G.; KAEHLER, A.: Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc. USA 2008, ISBN: 978-0-596-51613-0.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: doc. Ing. Kamil Říha, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato práce se zaměřuje na základní principy Houghovy transformace určené k detekci přímek, kružnic a obecných těles, jejich algoritmy, výpočetní náročnost a způsob přípravy obrazu pro detekci. Následný popis implementace jednotlivých algoritmů a vytvoření aplikace pro demonstraci jejich vlastností. K implementaci byl využit programovací jazyk MS Visual C++, volně dostupná knihovna OpenCV a uživatelské rozhraní bylo vytvořeno pomocí IDE Qt Creator 4.13.2 (Community).

KLÍČOVÁ SLOVA

Houghova transformace, Detekce přímek, Detekce kružnic, Detekce obecných těles, Rozpoznání objektů, Počítačové vidění, OpenCV, Gaussův filtr, Cannyho detektor hran

ABSTRACT

This thesis focuses on basic principles of Hough transform for the detection of lines, circles and arbitrary shapes, their algorithms, computational complexity and preparation of a picture for detection. The description of implementation of individual algorithms and creation of an aplication that demonstrates their properties. Implementation has been done in programming language MS Visual C++ with the help of open source library OpenCV and user interface has been created using IDE Qt Creator 4.13.2 (Community).

KEYWORDS

Hough transform, Lines detection, Circles detection, Detection of arbitrary shapes, Shape recognition, Computer vision, OpenCV, Gaussian filter, Canny edge detector

STEJSKAL, Jan. *Houghova transformace a její varianty*. Brno, 2020, 46 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Kamil Říha, Ph.D

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Houghova transformace a její varianty“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Kamilu Říhovi, Ph.D. za trpělivost, konstruktivní kritiku, cenné rady a podnětné návrhy při vývoji aplikace a psaní této práce.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Cílem této bakalářské práce je nastudovat teoretické principy Houghovy transformace a její různé varianty. Přesněji se bude jednat o detekci přímek, kružnic a obecných těles v obraze. Následně pak vytvořit aplikaci s implementovanými algoritmy detekce.

Houghova transformace (HT) byla patentována Paulem Houghem v roce 1962, jednalo se o algoritmus sloužící k detekci přímek v obraze. O několik let později, v roce 1972, ji Richard Duda a Petr Hart upravili do nynější podoby, kdy ji můžeme využít nejen k detekci přímek, ale také k detekci jiných jednoduchých obrazců jako jsou křivky, kružnice a elipsy. Pojmenovali ji klasická Houghova transformace.

Zobecněná Houghova transformace (Generalized Hough transform) byla představena Danem H. Ballardem v roce 1981. [5] Jedná se o modifikovanou verzi klasické Houghovy transformace založené na principu porovnávání předpřipravené šablony se vstupním obrazem. Zpopularizoval ji v komunitě počítačového vidění svým článkem „Generalizing the Hough transform to detect arbitrary shapes“. [4]

V dnešní době se Houghova transformace stále využívá v počítačovém a medicínském zpracování obrazu.

1 Klasická Houghova transformace

1.1 Detekce přímek

V minulosti a i nyní je stále opakujícím se problémem v počítačovém zpracování obrazu detekce přímých čar. V nejjednodušším případě obraz obsahuje řadu nespojitých, černých, obrazových bodů, které leží na bílém pozadí. Je zřejmé, že tento problém lze jednoduše vyřešit do požadované míry přesnosti tak, že budeme testovat čáry tvořené dvojicemi všech bodů. Tento přístup je však výpočetně náročný.

Rosenfeld přišel s metodou nahrazení původního problému nalezení bodů ležících na stejné přímce, matematicky ekvivalentním problémem nalezení souběžných křivek. Metoda spočívá v transformaci každého z obrazců na přímku v prostoru parametrů. [1] Hough využil známé parametry směrnice a a úseku b v rovnici 1.1. Bohužel jak směrnice, tak i úsek jsou neohraničené, což komplikuje použití této techniky.

$$y = ax + b. \tag{1.1}$$

1.1.1 Teorie detekce

Sada všech přímek v obrazové rovině může být reprezentována jedním bodem v parametrickém prostoru. Protože víme, že úsek a směrnice jsou neohraničené, využíváme tzv. normální parametrizaci. Tato parametrizace popisuje normálu přímky úhlem θ a její vzdáleností od středu souřadného systému ρ. [1] Rovnice přímky, která odpovídá této geometrii je

$$y = -\left(\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{\rho}{\sin\theta}\right). \tag{1.2}$$

Po troše úprav dostáváme

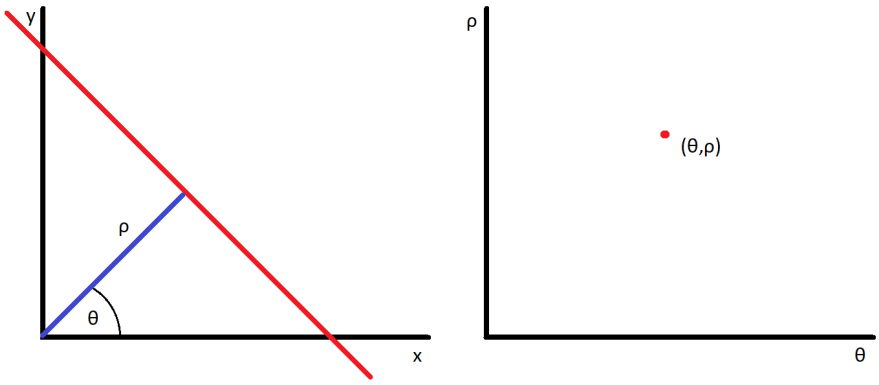
$$\rho = x\cos\theta + y\sin\theta. \tag{1.3}$$

Pokud omezíme θ na interval $[0, \pi]$, pak každá přímka bude mít své jedinečné parametry. S tímto omezením odpovídá každý řádek v rovině x, y jedinečnému bodu v rovině θ, ρ, tzv. Houghově rovině. [1]

Předpokládejme, že máme určitou sadu bodů $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ a chceme najít přímky, které jim vyhovují. Přetvoříme tedy body (x_i, y_i) na sinusoidy v Houghově rovině definované rovnicí

$$\rho = x_i\cos\theta + y_i\sin\theta. \tag{1.4}$$

Zjistíme, že křivky odpovídající obrazovým bodům ležícím na přímce mají společný průsečík. [1] Tento bod v Houghově rovině potom definuje přímku procházející danými body.



Obr. 1.1: Parametrický popis přímky.

Předpokládejme nyní, že máme body $\{(\theta_1, \rho_1), \ldots, (\theta_k, \rho_k)\}$ v Houghově rovině a všechny leží na křivce popsané rovnicí 1.3. Pak je možné jednoduše dokázat, že přímky v x-y rovině, které jsou tvořené těmito body, mají společný bod (x_0, y_0). [1]

Jednotlivé vlastnosti přeměny bodu na křivku je možné shrnout do těchto čtyř bodů:

1. Bod v rovině x-y odpovídá sinusové křivce v parametrické rovině.
2. Bod v parametrické rovině odpovídá přímce v rovině x-y.
3. Body ležící na stejné přímce v rovině x-y odpovídají křivkám se společným bodem v parametrické rovině.
4. Body ležící na stejné křivce v parametrické rovině odpovídají čarám procházejícím stejným bodem v rovině x-y. [1]

1.1.2 Algoritmus

Předpokládejme, že máme množinu bodů, tato množina tvoří n křivek v parametrickém prostoru, které se budou protínat právě v $n\frac{(n-1)}{2}$ bodech. [1] Principiálně pak lze nalézt podmnožinu těchto obrazových bodů, které leží na jedné přímce, nalezením shodných průsečíků v parametrické rovině. Tento přístup je však značně výpočetně náročný.

Výpočetní náročnost je však možné snížit. Podle Houghova základního návrhu určíme přijatelnou chybu v θ a ρ a omezíme Houghovu rovinu na oblast

$$0 \leq \theta < \pi,$$
$$-R \leq \rho < R.$$

Tato oblast bude potom považována za dvourozměrné pole, akumulátor. [1] Pro každý bod (x_i, y_i) v x-y prostoru se do pole ukládá odpovídající křivka, která je daná rovnicí 1.4. Ukládání probíhá inkrementací hodnoty v každé buňce podél křivky.

Daná buňka v akumulátoru tak zaznamená celkový počet křivek, které jí prochází. Po zpracování všech bodů obrazu se pole zkontroluje a naleznou se buňky s vysokým počtem průchozích křivek. Každá nalezená buňka pak udává, kolik obrazových bodů tvoří danou přímku.

1.1.3 Možnost specializace

Transformační techniku lze zobecnit a specializovat několika způsoby. Je vhodné podotknout, že je možné využít jakoukoliv techniku parametrizace. Jak již bylo zmíněno, Hough použil k parametrizaci směrnici a úsek. Tato parametrizace má však své nevýhody, které se projeví například tehdy, pokud bude ležet několik bodů na téměř svislé přímce. To způsobí, že hodnoty směrnice i úseku mohou být libovolně velké. Jak uvedl Rosenfeld, celý výpočet by se dal provést dvakrát a to výměnou x-ové a y-ové osy. To by však přineslo další komplikace. [1] Normální parametrizace se těmto nevýhodám vyhýbá.

Užitečnou vlastností této metody je schopnost detekce výskytu bodů obrazce na přímce, která má určitou specifikovanou vlastnost. Představme si, že chceme zjistit, zda významný počet bodů obrázku leží na přímce procházející bodem (x_0, y_0). Podle vlastnosti čtyři víme, že normální souřadnice každé takové přímky musí ležet na křivce

$$\rho = x_0 \cos\theta + y_0 \sin\theta.$$

Transformaci lze provádět obvyklým způsobem, ale pozornost může být omezena pouze na oblast Houghovi roviny v blízkosti této křivky. Pokud najdeme buňku s velikostí k v blízkosti této křivky, pak si můžeme být jistí, že k bodů leží na přímce procházející bodem (x_0, y_0) nebo alespoň v jeho blízkosti. Podobně, pokud máme zájem pouze o přímky, které mají daný směr, omezíme naši pozornost na část Houghovi roviny v blízkosti θ_0. [1]

1.2 Detekce kružnic

Každý den se setkáváme s mnoha předměty, které obsahují kruhové prvky. Detekce těchto objektů v digitálním obraze je důležitá pro analýzu obrazu v různých aplikacích počítačového vidění.

1.2.1 Parametrizace

Kruh je v parametrickém prostoru jednodušší znázornit než přímku, protože parametry kružnice mohou být přímo přeneseny do parametrického prostoru. Rovnice

kruhu je

$$(x-a)^2 + (y-b)^2 = r^2. \tag{1.5}$$

Jak je vidět, kruh má tři parametry r, a a b. Kde a a b udávají střed kružnice a kde r značí její poloměr. Parametricky je kruh reprezentován jako

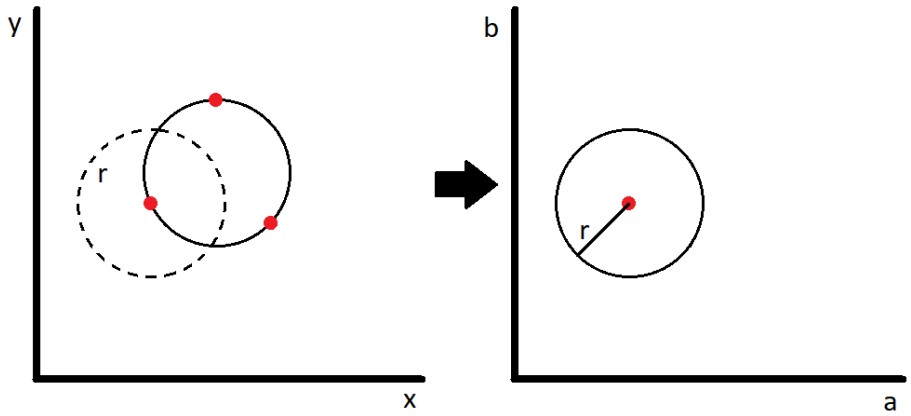
$$b = x_0 + r\cos\theta, \tag{1.6}$$

$$a = y_0 + r\sin\theta. \tag{1.7}$$

Houghův prostor pro detekci kružnic tak bude patřit R^3, zatímco pro přímku patřil pouze do R^2. [2] S rostoucím počtem parametrů potřebných k popisu tvaru a s rostoucím rozměrem Houghova prostoru, roste i výpočetní náročnost Houghovy transformace. Z tohoto důvodu je HT obecně zvažována pouze pro jednoduché tvary. Pro zjednodušení parametrické reprezentace kruhu lze na poloměr pohlížet jako na konstantu, popřípadě lze algoritmus omezit na určitý počet poloměrů.

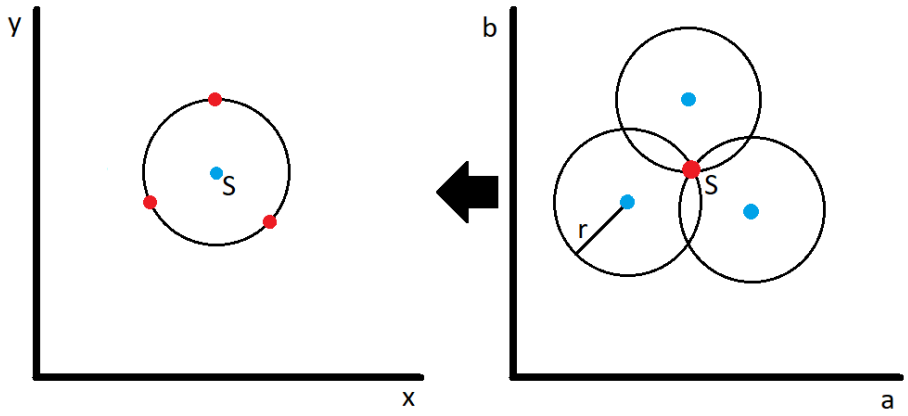
1.2.2 Princip detekce a akumulátor

Víme, že vzdálenost středu S kružnice k jejím bodům b je vždy stejná, jako vzdálenost těchto bodů ke středu jejich kružnice. Tuto vzdálenost označíme jako poloměr r_0. Pokud pro všechny body b vykreslíme do Houghova prostoru kružnice o poloměru r_0, kdy bod b_i bude udávat střed této nové kružnice. (obr. 1.2) Zjistíme, že se všechny tyto nové kružnice protnou v jednom bodě. (obr. 1.3) Jedná se o bod S, který nám udává střed původní kružnice.



Obr. 1.2: Vykreslování do Houghova prostoru.

V praxi potom pouze inkrementujeme akumulátor na pozici bodů ležících podél všech vykreslovaných kružnic. V místech, kde se kružnice v Houghově prostoru protnou, se v akumulátoru vyskytne lokální maximum. To nám určí souřadnice hledaného středu.



Obr. 1.3: Vyhledání středu kružnice.

1.2.3 Náročnost na výpočet

Při běžném způsob implementace bude výpočetní náročnost odpovídat přibližně n^3. To pro běžné využití není dostačující. Naštěstí můžeme tuto výpočetní náročnost určitými operacemi snížit.

Nejjednodušším způsobem je omezení počtu testovaných poloměrů r. Kdyby jsme si například vzali obrázek o rozměrech 500×500 pixelů, tak by bylo nerozumné vyhledávat kružnice pro všech 500 poloměrů. Největší kružnice, která by se mohla v obrázku nacházet bude mít poloměr o velikosti 250 pixelů. Ale jelikož je pravděpodobnost výskytu takovéto kružnice v obrázcích či fotografiích velmi malá, můžeme počet hledaných poloměrů snížit o (přibližně) 30 pixelů. To samé by se dalo říct o kružnici s poloměrem 20 pixelů a menším. Pravděpodobnost existence takové kružnice je velmi malá a můžeme tedy testované poloměry omezit na rozsah

$$20 \leq r < 220.$$

Snížíme tak výpočetní čas o celých 60%.

Jedním z možných způsobů snížení výpočetní náročnosti je nalezení hran ve vstupním obraze před provedením samotného výpočtu. Počet hran tvoří obvykle pouze 1-2% celkového počtu pixelů v obraze. Pojmenujme si tedy tento počet hran v obraze jako h, potom výpočetní náročnost klesne z n^3 na $n^2 + r \times h$.

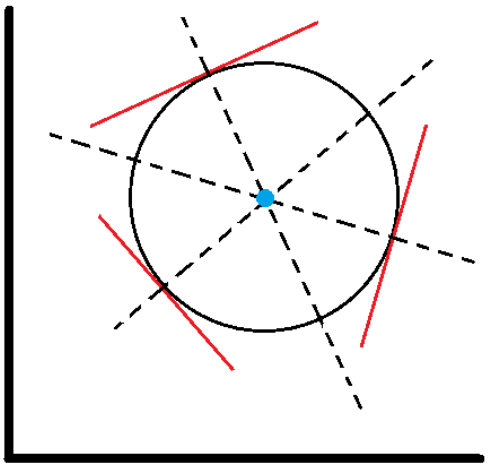
Dalším z technik je snížení počtu inkrementovaných polí pro každou z kružnic. Víme, že kružnice má 360 °. Pokud jsou detekované kružnice ve fotografii dobře osvětlené a mají vysoký kontrast vůči jejich pozadí, můžeme předpokládat, že dojde k nalezení velkého počtu hran. V takovém případě není nutné inkrementovat všech 360 polí pro každou z kružnic, můžeme tento počet omezit na polovinu, či třetinu a získat dvojnásobné nebo trojnásobné zrychlení algoritmu.

1.2.4 Metoda Houghova gradientu

V původním algoritmu jsme vykreslovali kružnice kolem hran a zkoumali, zdali se tyto kružnice protaly v jednom bodě. Pokud ano, nalezli jsme střed a poloměr hledané kružnice. Tato metoda je však výpočetně náročnější, jelikož jsme se pohybovali v prostoru R^3. Metoda Houghova gradientu využívá informace o sklonu hrany ke zrychlení tohoto výpočtu. Skládá se ze dvou hlavních kroků:

- Nalezení vhodných kandidátů pro střed kružnice.
- Nalezení poloměru, který nejlépe odpovídá původní kružnici. [3]

Postup implementace je takový, že se nejdříve naleznou hrany s využitím Cannyho detektoru hran a pak se pomocí Sobelova operátoru zjistí hodnota gradientu. Dále se pro každou nalezenou hranu inkrementují všechny body ležící v obou směrech gradientu dané hrany. (obr. 1.4) Následně se prohledá akumulátor a vyberou se potenciální středy hledaných kružnic, jejichž hodnota je vyšší než určená hranice. [3]



Obr. 1.4: Princip detekce potenciálních středů – Houghův gradient

Po nalezení všech potenciálních středů a jejich uložení do jednorozměrného pole, se pro každý z těchto středů vypočítá vzdálenost od detekovaných hran a inkrementují se příslušné buňky akumulátoru poloměrů. [3] Dále se akumulátor prohledá. Nalezne se poloměr, který nejlépe odpovídá detekované kružnici a ten se uloží. Všechny duplicitní a velmi blízké středy o podobných poloměrech se vyřadí a zbylé středy jsou považovány za hledané kružnice.

2 Zobecněná Houghova transformace

V obraze jsou důležité informace o objektu velmi často obsaženy ve tvaru jeho hran. Experimenty prováděné na lidském zrakovém systému prokázaly, že hrubé vyznačení hran často postačuje k rozpoznávání objektů. Bude se tedy jednat o popis obecného algoritmu pro detekci objektů, které mají svůj specifický tvar, v obraze detekovaných hran. V tomto ztvárnění body obrazu neobsahují informace o úrovni šedi, ale informace o síle a orientaci změny úrovně šedi. [4]

Původní Houghova transformace nevyužívala orientační informace o hraně, jelikož jsme byli schopni hledané přímky a křivky parametricky popsat. To je však u komplexního objektu složeného z velkého množství křivek o různém poloměru velice složité. Houghův zobecněný algoritmus využívá získané informace k definování orientace hrany a její mapování na referenční bod objektu.

2.1 Teorie transformace

Pro zobecněný tvar $\mathbf{a}$ jsou definovány tyto parametry:

$$\mathbf{a} = \{\mathbf{y}, \mathbf{s}, \theta\}, \tag{2.1}$$

kde $\mathbf{y} = (x_r, y_r)$ je referenční – středový bod objektu, $\mathbf{s} = (s_x, s_y)$ jsou dva vektory popisující změnu měřítka a θ úhel popisující jeho orientaci. Umístění středového bodu $\mathbf{y}$ je popsáno pomocí tabulky obsahující orientace a vzdálenosti všech hran. Parametry $\mathbf{s}$ a θ se následně vypočítají jako transformace této tabulky. [4]

Merlin a Farber popsali ve své práci jak využít Houghův algoritmus k detekci křivek, které nelze analyticky popsat. Každý obrazec musí mít svůj specifický vztažný bod. To nám umožňuje použít následující algoritmus k detekci tvaru, který má své hrany $\mathbf{x_b}$, vztažené k nějakému referenčnímu bodu středu $\mathbf{y}$. [4]

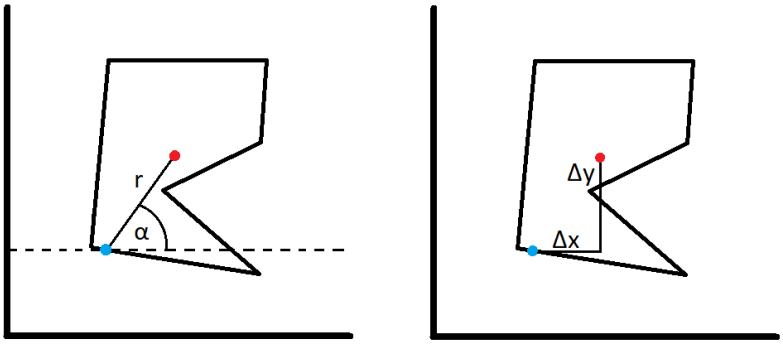
Merlin-Farberův algoritmus je však nepraktický pro reálná obrazová data. Na obrázku s velkým množstvím detekovaných hran bude mnoho falešných instancí požadovaného tvaru kvůli náhodnému uspořádání pixelů. Nicméně je to logický předchůdce zobecněného Houghova algoritmu. Klíčem k zobecnění Houghova algoritmu na libovolné tvary je využití směrových informací. Směrové informace kromě toho, že algoritmus dělají rychlejší také výrazně zlepšují jeho přesnost.

2.1.1 R-table

R-table, nebo také LuT (Look-up table) je snadno zkonstruovatelný zkoumáním hran objektu. Jedná se tedy o tabulku, která popisuje tvar hledaného objektu. Konstrukce této tabulky je provedena následovně.

Zvolíme referenční bod $\mathbf{y}$ zkoumaného objektu. Pro každou hranu $\mathbf{x}$ vypočítáme směr přechodu ϕ a vzdálenost středu od hrany $\mathbf{r} = \mathbf{y} - \mathbf{x}$. Každé $\mathbf{r}$ následně uchováme jako funkci ϕ. Vypočtené hodnoty jsou vektory, proto může mít jedna hodnota úhlu ϕ uloženo mnoho hodnot $\mathbf{r}$. [4] Tabulka je využívána k detekci instancí objektu S v obraze následujícím způsobem.

Pro každou hranu $\mathbf{x}$ s určitou hodnotou sklonu ϕ, která se nachází v obraze, inkrementujeme všechny odpovídající body v akumulátoru A, které se nachází na pozici $\mathbf{x} + \mathbf{r}$. Za předpokladu, že $\mathbf{r}$ je hodnota z tabulky indexovaná podle ϕ. Maxima v A odpovídají možným instancím hledaného objektu S. [4]



Obr. 2.1: Ukládání hodnot do LuT.

2.1.2 Vlastnosti tabulky R-table

Do této chvíle jsme brali v potaz pouze objekty s pevnou orientací a měřítkem. Pro nalezení objektů s libovolnou orientací θ a měřítkem $\mathbf{s}$ musíme nejdříve přidat tyto nové parametry do popisu tvaru. Akumulátor se nyní skládá ze čtyř rozměrů odpovídajících parametrům $(\mathbf{y}, \mathbf{s}, \theta)$.

Vytvoříme konkrétní tabulku R pro objekt S. Provedením jednoduché transformace této tabulky jsme schopni detekovat zmenšené nebo otočené instance stejného objektu. Například pokud je tvar zmenšen o s a tuto operaci označíme jako T_s, pak

$$T_s[R(\phi)] = sR(\phi), \tag{2.2}$$

všechny vektory jsou zmenšeny o s. A pokud je objekt otočen o úhel θ a operace otočení je označena jako T_θ, pak

$$T_\theta[R(\phi)] = \mathrm{Rot}\{R[(\phi - \theta) \bmod 2\pi], \theta\}, \tag{2.3}$$

všechny indexy se zmenší o θ modulo 2π, najdou se příslušné vektory $\mathbf{r}$ a ty se pak pootočí o úhel θ. [4]

Další užitečnou vlastností je změna referenčního bodu. Pokud chceme zvolit nový referenční bod $\mathbf{y}'$, zjistíme jeho vzdálenost $\mathbf{r}$ od původního referenčního bodu podle $\mathbf{y} - \mathbf{y}' = \mathbf{r}$ a tento rozdíl přičteme ke všem hodnotám tabulky $R(\phi) + \mathbf{r}$. [4]

2.1.3 Složené objekty

Pokud máme složený objekt S, který má dvě části S_1 a S_2, pak je tento objekt možné nalézt s využitím tabulek jednotlivých částí R_1 a R_2. Pokud $\mathbf{y}, \mathbf{y_1}, \mathbf{y_2}$ budou referenční body objektu S a jeho částí S_1 a S_2, potom můžeme vypočítat $\mathbf{r_1} = \mathbf{y} - \mathbf{y_1}$ a $\mathbf{r_2} = \mathbf{y} - \mathbf{y_2}$. Složená tabulka $R_s(\phi)$ pak bude definována jako

$$R_s(\phi) = [R_{s_1}(\phi) + \mathbf{r_1}] \cup [R_{s_2}(\phi) + \mathbf{r_2}], \tag{2.4}$$

to znamená, že pro každý index s hodnotou ϕ je $\mathbf{r_1}$ přičteno k $R_{s_1}(\phi)$ a $\mathbf{r_2}$ je přičteno k $R_{s_2}(\phi)$. Jejich sjednocení je následně uloženo v $R_s(\phi)$. [4]

Podobně můžeme definovat objekty jako rozdíl mezi tabulkami s podobnými záznamy

$$R_s = R_{s_1} - R_{s_2}. \tag{2.5}$$

Objekt S bude definovaný tvary S_1 a S_2, ale jejich společné hodnoty budou vymazány. Primární využití operace sjednocení je detekovat objekty, které jsou složeny z jednodušších tvarů. Nicméně, možnost využití rozdílu má také užitečnou funkci. Jejím užitím mohou být zkonstruovány tabulky, které rozlišují mezi dvěma podobnými objekty. [4]

Zatímco rovnice 2.4 je jedním ze způsobů aplikace Houghovy transformace k detekci složených objektů, nemusí to být nejlepší způsob. Volba referenčního bodu může významně ovlivnit přesnost transformace. Vzhledem k tomu, že středový bod se při zvětšení od objektu vzdaluje, mohou malé úhlové chyby v ϕ způsobit velké chyby ve vektorech $R(\phi)$.[4]

2.2 Způsoby inkrementace

Pokud použijeme metodu navyšování pole akumulátoru určitou hodnotou, pak obsah akumulátoru bude přibližně úměrný obvodu tvaru, který je zjistitelný na obrázku. Tato metoda je vhodná k hledání objektů, které mají velkou část obvodu viditelnou.

V závislosti na kvalitě obrazových dat je k dispozici několik různých metod inkrementace akumulátoru. Pokud jsou detekovány krátké části obvodu, jak by tomu mohlo být v případě, že je objekt částečně zakrytý, pak by mohla být vhodnější metoda inkrementace o absolutní hodnotu gradientu

$$\mathbf{A}(\mathbf{a}) = \mathbf{A}(\mathbf{a}) + |\, g(\mathbf{x}) \,|. \tag{2.6}$$

Samozřejmě, že můžeme tyto metody zkombinovat, potom bychom inkrementovali i o určitou hodnotu c

$$\mathbf{A}(\mathbf{a}) = \mathbf{A}(\mathbf{a}) + \mid g(\mathbf{x}) \mid + c, \tag{2.7}$$

kde c je konstanta. [4]

Další možností je použití informací o lokálním zakřivení objektu. Pomocí této metody jsou zkoumány pixely sousedních hran pro výpočet přibližného zakřivení. Tato metoda však výrazně komplikuje tabulku. Nyní spolu s každou hodnotou r musí být uloženy i odpovídající hodnoty zakřivení. S přírůstky lokální křivky potom inkrementujeme akumulátor takto:

$$\mathbf{A}(\mathbf{a}) = \mathbf{A}(\mathbf{a}) + K, \tag{2.8}$$

kde K je hodnota přibližného zakřivení. [4]

Při hledání složeného objektu mohou mít různé části různou důležitost. To můžeme snadno zařídit přiřazením důležitosti w_i každé tabulce R_{s_i} tak, že pro každý záznam v R_{s_i}, inkrementujeme o násobek w_i místo jednotkové hodnoty. [4]

3 Příprava obrazu pro detekci

Před použitím algoritmu je třeba připravit vstupní obraz. Postup a některé z metod, které je možné použít pro jednotlivé kroky, budou popsány v této kapitole.

3.1 Převod na stupně šedi

Tento krok je důležitý, jelikož téměř všechny programové implementace detekce hran nepřijímají barevný obraz jako validní. (obr. 3.3a)

3.1.1 Intenzita a reprezentace barev

Obraz ve stupních šedi je vyjádřen jednou hodnotou intenzity. Ta může být reprezentována hodnotami 0 % pro absolutní černou, 100 % pro absolutní bílou barvu a hodnotami mezi nimi. Tato notace se používá v akademických pracích, ale nedefinuje co je „černá“ nebo „bílá“ z hlediska kolorimetrie. Někdy stupnici obracíme. Toho se využívá například při tisku. Potom velikost intenzity označuje kolik inkoustu je použito.

Ikdyž je možné s barvami pracovat jako s racionálními čísly, ve výpočetní technice se hodnoty barev definují jako celá nezáporná čísla. Jak hodnoty barev, tak hodnoty intenzity jsou obvykle uloženy jako 8-mi bitová čísla. Těchto 8 bitů nám dovoluje vytvořit 256 různých odstínů.

3.1.2 Převod na stupně šedi z RGB kanálů

Běžnou metodou je použití principů kolorimetrie pro výpočet hodnot stupňů šedi tak, aby měly stejnou svítivost (relativně stejnou) jako původní barevný obrázek. Kromě stejné svítivosti tato metoda také zajišťuje, že oba snímky budou mít při zobrazení stejnou absolutní svítivost.

Chceme-li převést barvu z barevného prostoru na znázornění jeho jasu ve stupních šedi, musí být obraz nejprve linearizován, aby bylo možné na lineární barevné složky použít příslušný vážený součet pro výpočet lineární svítivosti.

Existuje velké množství modelů, podle kterých by jsme mohli převést obraz do stupňů šedi. Liší se výhradně vahou, kterou přidělují jednotlivým R, G, B složkám obrazu. Jedním z prvních modelů byl

$$Y = 0,33R + 0,33G + 0,33B,$$

ten však není vhodný, protože lidské oko je jinak citlivé na každý z barevných kanálů. Nejčastěji se využívá model

$$Y = 0,229R + 0,587G + 0,114B,$$

v obou případech je Y jasová složka a R, G, B jsou jednotlivé barevné složky obrazu. [12]

3.2 Aplikace rozmazání – vyhlazení

Dále se doporučuje aplikovat na černobílou fotografii některou z dostupných forem rozmazání – blur. Jedná se o způsob odstranění šumu z obrazu. (Obr. 3.3b)

3.2.1 Gaussův filtr

Gaussův filtr je lineární typ okénkového filtru. Byl nazván po slavném vědci Carlu Gaussovi, jelikož se při výpočtu ve filtru využívá Gaussova rozdělení – funkce, kterou používal ve své tvorbě. [6] Jiný název pro tento filtr je Gaussovo rozostření.

Při využití tohoto filtru dochází ke konvoluci obrazu Gaussovou funkcí a potlačují se vysoké frekvence v obrázku. Můžeme tedy říct, že se jedná o nízkofrekvenční filtr. Jak již bylo řečeno, Gaussův filtr využívá Gaussovo rozdělení. To je vyjádřeno rovnicí

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-a)^2}{2\sigma^2}}, \tag{3.1}$$

kde a nám určuje pozici na křivce. Můžeme však výraz zjednodušit a zafixovat hodnotu a na nula. [6] Potom dostáváme rovnici jednorozměrné Gaussovy funkce

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}. \tag{3.2}$$

Pro vyhlazení obrázku však potřebujeme dvourozměrnou Gaussovu funkci, toho můžeme dosáhnout provedením jednorozměrné funkce ve dvou směrech

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}, \tag{3.3}$$

kde x je vzdálenost od počátku x-ové osy, y je vzdálenost od počátku y-ové osy a σ je standardní rozložení Gaussovy distribuce.[7] Při použití této dvourozměrné funkce dostáváme obraz, který je tvořen soustřednými kružnicemi s Gaussovým rozdělením od středového bodu.

Hodnoty získané tímto rozdělením se použijí k vytvoření konvoluční matice, kterou aplikujeme na původní obrázek. Na obr. 3.1 můžeme vidět příklad takovéto matice, je nutno podotknout, že hodnoty v matici obvykle bývají racionální čísla, ale pro ukázku byla volena celá čísla. Každý pixel potom nabude novou hodnotu, kterou získáme vypočítáním váženého průměru tohoto pixelu a jeho okolí. Hodnotě původního pixelu je přiřazena nejvyšší váha a sousedním pixelům nižší v závislosti na vzdálenosti od středu tabulky. To má za následek rozostření, které zachovává hrany lépe než jiné, jednotnější rozostřovací filtry. [7]

1	1	2	1	1
1	4	8	4	1
2	8	16	8	2
1	4	8	4	1
1	1	2	1	1

Obr. 3.1: Přiklad konvoluční matice.

Aplikace několika Gaussových filtrů na obraz má stejný účinek jako aplikace jednoho Gaussova filtru, jehož rozměry jsou dány odmocninou součtu druhých mocnin použitých rozměrů filtru. Pokud bychom například na obraz aplikovali Gaussův filtr s rozměry 5 a 12, dostali bychom stejný výsledek jako při aplikaci filtru s poloměrem 13, protože $\sqrt{5^2 + 12^2} = 13$. [7] Ze vztahu vyplývá, že výpočet Gaussova rozostření nelze urychlit aplikací většího počtu filtrů o menších rozměrech.

3.3 Detekce hran

Dalším krokem je použití některého z dostupných algoritmů detekce hran. (Obr. 3.3c)

3.3.1 Sobelův operátor

Sobelův operátor, někdy také nazývaný Sobel-Feldmanův operátor nebo Sobelův filtr je pojmenován po Irwinu Sobelovi a Garym Feldmanovi. Sobel a Feldman představili svoji práci „Izotropic 3x3 Image Gradient Operator“ v roce 1968. [8]

Sobelův operátor je velmi podobný Prewittové operátoru. Jedná se také o derivační masku a používá se k detekci hran v horizontálním a vertikálním směru. Hlavním rozdílem mezi těmito operátory je, že koeficienty Sobelovy masky nejsou pevně dané a je je možné upravit podle potřeby. [9] Podmínkou však je, že neporušíme žádnou z daných vlastností derivačních masek.

Na obrázcích 3.2 můžeme vidět podobu Sobelových masek v x-ovém a y-ovém směru. Tyto masky fungují stejně jako operátor Prewittové s jediným rozdílem a to, že Sobelův operátor má hodnoty v druhé a osmé buňce v horizotálním směru a čtvrté a šesté buňce ve vertikálním směru, rovny hodnotám -2 a 2.

Pokud aplikujeme tuto masku na libovolný vstupní obraz, dostaneme výstupní obraz obsahující detekované hrany ve vertikálním nebo horizontálním směru. Celá tato operace funguje jako derivace prvního řádu, počítají se tedy rozdíly intenzity pixelů v oblasti hrany. Z toho můžeme usoudit, že Sobel-Feldmanův operátor je

-1	0	1
-2	0	2
1	0	1

(a) Maska ve směru y

-1	-2	-1
0	0	0
1	2	1

(b) Maska ve směru x

Obr. 3.2: Sobelův operátor

relativně nenáročný z hlediska výpočtů. Na druhou stranu aproximace gradientu, kterou vytváří, je poměrně hrubá.

3.3.2 Cannyho detektor hran

Cannyho detektor hran byl vyvinut Johnem F. Cannym v roce 1986, jedná se o vícestupňový algoritmus pro detekci široké škály hran v obraze. Teorii, na které je Cannyho detektor hran založen, popsal ve své práci nazvané „Computational theory of edge detection“ (Výpočetní teorie detekce hran). [11]

Cannyho detektor hran se široce uplatňuje v oblasti počítačového vidění. Canny zkoumal požadavky na detekci hran v různých systémech a zjistil, že jsou velice podobné. Obecná kritéria pro detekci hran jsou:

1. Detekce by měla co nejpřesněji zachytit hrany zobrazené na obrázku.
2. Bod hrany nalezený operátorem by se měl zobrazit přesně na středu hrany.
3. Určená hrana obrázku by měla být označena pouze jednou hranou a pokud je to možné, šum obrazu by neměl vytvářet falešné hrany. [11]

Mezi všemi vyvinutými metodami detekce hran je Cannyho algoritmus nejoblíbenější a nejrozšířenější. Může za to jeho schopnost optimálně splnit všechny tři jmenovaná kritéria a jeho vysoká spolehlivost.

Celý proces detekce jde rozdělit do 5 kroků:

1. Aplikace Gaussova filtru pro vyhlazení obrazu.
2. Nalezení velikosti natočení pixelu pomocí Sobelova operátoru.
3. Proces tenčení nalezených hran.
4. Použití dvojitého prahu pro určení potenciálních hran.
5. Potlačení falešných hran.

Funkce a princip Gaussova filtru byly popsány v sekci 3.2.

Jak již bylo zmíněno v sekci (3.3.1), Sobelův operátor provádí derivaci prvního řádu. Této jeho vlastnosti můžeme využít pro zjištění orientace hrany. Samotný

výpočet je velice jednoduchý

$$\theta = \arctan \frac{G_y}{G_x}, \tag{3.4}$$

kde θ je úhel natočení hrany, G_x a G_y je první derivace podle x a y.

Proces tenčení je aplikován na místa s ostrými změnami v intenzitě. Postupuje se podle dvou základních kroků. Porovnají se síly hrany aktuálního pixelu se silou hrany pixelu v kladném a záporném směru sklonu. A pokud je síla hrany aktuálního pixelu vyšší než síla porovnávaných pixelů, zůstane zachován. V opačném případě je jeho hodnota potlačena.

Po aplikaci tenčení poskytují nalezené okrajové pixely přesnější znázornění reálných okrajů v obrázku. Hran vytvořených šumem se zbavíme volbou dvou prahů o různých velikostech. Pokud hodnota pixelu překročí vyšší práh je označen jako silná hrana. Pokud se nachází mezi vyšším a nižším prahem, je označen za slabou hranu a v případě, že nepřekročí nízký práh, je jeho hodnota potlačena.

V poslední fázi je třeba rozlišit falešné slabé hrany od těch pravých. Slabá pravá hrana je obvykle připojena k silné hraně, zatím co falešná hrana způsobená šumem není. Dochází tedy ke zkoumání okolí každé slabé hrany a pokud se v jejím osmiokolí nachází alespoň jedna silná hrana, je tento pixel zachován. Ostatní slabé hrany, které nesplňují tuto podmínku jsou potlačeny.

(a) Stupně šedi. (b) Aplikované rozmazání. (c) Detekované hrany.

Obr. 3.3: Ukázka připravené fotografie.

4 Implementace algoritmů

4.1 Implementace detekce přímek

1. Pro ukázku algoritmu detekce jsem si vybral tuto fotografii ohrady. (obr. 4.1).

Obr. 4.1: Detekce přímek – zvolená fotografie

2. Na fotografii byl aplikován postup předzpracování, o kterém se mluvilo v sekci 3. Pro převod na stupně šedi byla využita funkce *cvCtvColor()*, která vyžaduje určitý výstupní formát. Jelikož potřebuji fotografii ve stupních šedi, zvolím *COLOR_BGR2GRAY*. Dále jsem využil funkci *GaussianBlur()*, která pomocí Gaussova konvolučního jádra vyhladí vstupní obraz. Posledním krokem předzpracování je detekce hran, k tomu jsem použil funkci *Canny()*. Všechny tyto funkce jsou obsaženy v knihovně *OpenCV*. Na obrázku 4.2 se nachází výsledný binární obraz detekovaných hran.
3. Předpřipravím si dvě pole, ta se následně naplní hodnotami sinů a cosinů úhlů v rozsahu od $\langle -90\,°, 90\,° \rangle$. Jelikož jsou funkce *sin()* a *cos()* značně pomalé, vypočítání těchto hodnot předem výrazně zrychlí algoritmus.

```
float theta = 0, inc = M_PI / 180;
for(int i = 0; i < 180; i++) {
    CosValues[i] = cos(theta - M_PI_2);
    SinValues[i] = sin(theta - M_PI_2);
    theta += inc;
}
```

Výpis 4.1: Před vypočítání hodnot úhlů

Obr. 4.2: Detekce přímek – detekované hrany ve fotografii

4. Inicializuje se dvourozměrný akumulátor parametrického prostoru pro ukládání vykreslovaných křivek. Rozměry pole budou $(2 \times c, 180)$, kde c je délka úhlopříčky fotografie. Ta je vypočítána pomocí Pythagorovy věty, a potom bude výška a b šířka fotografie.

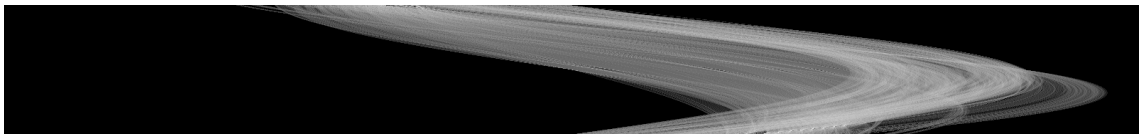
$$c = \sqrt{a^2 + b^2},$$

5. Algoritmus bude procházet binární obraz detekovaných hran. V momentě, kdy narazí na hranu, tj. bílý pixel s hodnotou jasu 255, vypočítá velikost ρ podle rovnice 1.4 a inkrementuje akumulátor na pozici (θ, ρ) o jedna. Výpočet se provede pro každé θ nabývající hodnot od $\langle -90\,°, 90\,° \rangle$ a celý tento postup se opakuje, dokud není prohledán celý obraz.

```
for(int i = 0; i < FoundEdges->rows; i++) {
    for(int j = 0; j < FoundEdges->cols; j++) {
        if(FoundEdges->at<uchar>(i,j) == 255) {
            for(int theta = 0; theta < 180; theta++) {
                rho = cvRound(j * CosValues[theta] +
                              i * SinValues[theta] +
                              MaxLength);
                Accumulator->at<int>(theta, rho)++;
            }
        }
    }
}
```

Výpis 4.2: Naplnění akumulátoru

Po dokončení této části detekce vypadá naplněný akumulátor jako na obr. 4.3 (Obrázek byl otočen o 90 °).



Obr. 4.3: Detekce přímek – příklad naplněného akumulátoru

6. Následně dojde k prohledání naplněného akumulátoru a vyhledání všech lokálních maxim (míst, která překročila určený práh). Každý takový bod je algoritmem považovat za přímku. Algoritmus zjistí hodnoty ρ a θ a provede zpětnou transformaci z Houghovy roviny do kartézského souřadného systému podle rovnic

$$x_{1,2} = \rho \cos\theta \pm m \sin\theta,$$
$$y_{1,2} = \rho \sin\theta \pm m \cos\theta,$$

kde m je offsetová hodnota. Tyto dva získané body, které jsou od sebe dostatečně vzdálené, aby odchylka při vykreslování byla malá, jsou využity k vyznačení nalezené přímky ve výsledné fotografii červenou čarou.

```
Scalar red(0, 0, 255);
for(int i = 0; i < Accumulator->rows; i++) {
    for(int j = 0; j < Accumulator->cols; j++) {
        if(Accumulator->at<int>(i, j) >= Treshold) {
            rho = j - MaxLength;
            theta = (i - 90) * CONVRAD;
            float a = cos(theta), b = sin(theta);
            float x0 = a * rho, y0 = b * rho;
            Point pt1(cvRound(x0 + 10000 * (-b)),
                      cvRound(y0 + 10000 * (a))),
                  pt2(cvRound(x0 - 10000 * (-b)),
                      cvRound(y0 - 10000 * (a)));
            line(*Result, pt1, pt2, red, 2, LINE_AA);
        }
    }
}
```

Výpis 4.3: Prohledání akumulátoru a vykreslení přímek do obrazu

7. Po detekci jsou ve fotografii vyznačené přímky, které jsou tvořeny prkny ohrady. Pro tento výsledek byl práh nastaven na hodnotu 170. (Obr. 4.4)

Obr. 4.4: Detekce přímek – nalezené přímky v obraze

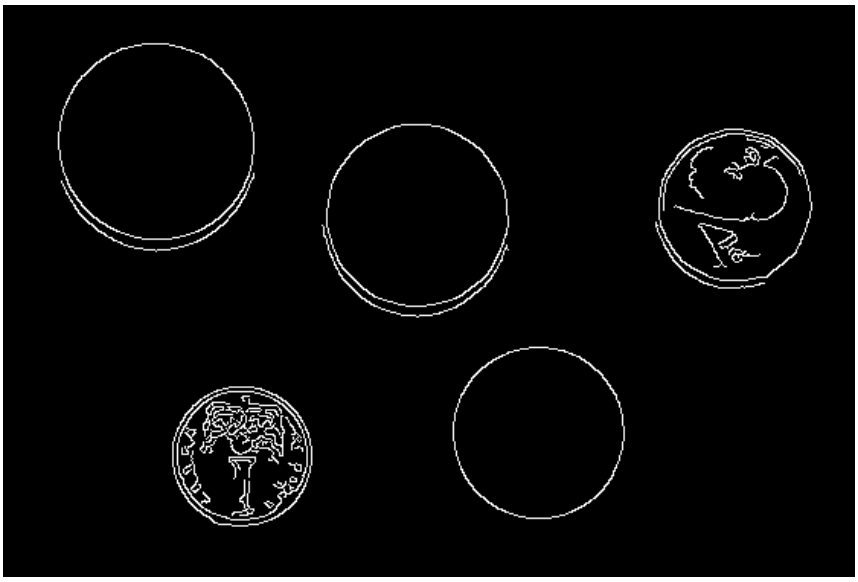
4.2 Implementace detekce kružnic

1. Pro ukázku jsem si vybral tuto fotografii několika mincí o různých poloměrech. (obr. 4.5)

Obr. 4.5: Detekce kružnic – zvolená fotografie

2. Na zvolenou fotografii je uplatněn postup předzpracování, o kterém jsem hovořil v sekci 3. Pro převod na stupně šedi byla použita funkce *cvCtvColor()*, která jako jeden ze vstupních parametrů požaduje typ formátu výstupní fotografie. Jelikož

potřebuji obraz ve stupních šedi, zvolím *COLOR_BGR2GRAY*. Pro vyhlazení využiji *GaussianBlur()* s konvoluční maticí o rozměrech 5×5. Dalším a posledním krokem je aplikování některé z dostupných metod detekce hran. Použiji funkci *Canny()*, která provádí, jak již název napovídá, Cannyho detekci hran. Všechny tyto funkce jsou obsaženy v knihovně *OpenCV*. Po dokončení těchto kroků dostávám potřebný binární obraz detekovaných hran, který je možné vidět na obrázku 4.6.



Obr. 4.6: Detekce kružnic – detekované hrany ve fotografii

3. Jedním ze způsobů zrychlení detekce je vypočítání hodnoty sinů a cosinů úhlů předem a to proto, že operace *sin()* a *cos()* jsou velmi pomalé. Vytvořením tabulky, do které bude algoritmus pouze nahlížet, získám značné výpočetní zrychlení. Počet těchto úhlů a tedy i bodů, které budou vykreslovány do akumulátoru pro každou hranu, se pohybuje v rozmezí od 20 do 180. V kódu je udáván proměnnou *pointsOnCircle*.

```
float theta = 0, inc = 2*M_PI/pointsOnCircle;
for (int i = 0; i < pointsOnCircle; i++) {
    CosValues[i] = cos(theta);
    SinValues[i] = sin(theta);
    theta += inc;
}
```

Výpis 4.4: Před vypočítání hodnot úhlů

4. Dalším z implementovaných způsobů jak algoritmus urychlit je nalezení a uložení souřadnic hran v obraze předem. Dojde tak ke značnému snížení počtu pixelů, které algoritmus musí projít pro každý testovaný poloměr. Jelikož se počet hran

v obraze obvykle pohybuje kolem 1-2 %, bude provedením tohoto kroku získáno značné výpočetní zrychlení.

```
vector<Point> VectorOfFoundEdges;
for(int i = 0; i < FoundEdges->rows; i++) {
    for(int j = 0; j < FoundEdges->cols; j++) {
        if(FoundEdges->at<uchar>(i, j) == 255) {
            VectorOfFoundEdges.push_back(Point(j, i));
        }
    }
}
```

Výpis 4.5: Nalezení a uložení všech hran ve fotografii do vektoru

5. Jelikož budu chtít obrazy akumulátorů pro jednotlivé poloměry ukládat a následně zobrazit, vytvořím si dynamické pole, do kterého budou naplněné akumulátory postupně přidávány.
6. Nyní k samotné detekci. Algoritmus si pro každý testovaný poloměr vytvoří nový akumulátor o rozměrech vstupní fotografie. Do tohoto nového akumulátoru se následně, pro všechny předem nalezené hrany, vykreslí kružnice o poloměru r se středem na pozici dané hrany. Souřadnice jednotlivých bodů na vykreslované kružnici vypočítá podle rovnic

$$y = y_0 - r\cos\theta,$$
$$x = x_0 - r\sin\theta,$$

kde r je poloměr kružnice, (x_0, y_0) souřadnice středu a (x, y) jsou souřadnice bodu na kružnici.

```
Mat Accumulator = Mat::zeros(FoundEdges->rows,
                             FoundEdges->cols,
                             CV_8U);
for (Point edge: VectorOfFoundEdges) {
    for(int i = 0; i < pointsOnCircle; i++) {
        y = edge.y - cvRound(r * CosValues[i]);
        x = edge.x - cvRound(r * SinValues[i]);
        if(y >= 0 && y < Accumulator.rows &&
           x >= 0 && x < Accumulator.cols)
            Accumulator.at<uchar>(y, x)++;
    }
}
```

Výpis 4.6: Naplnění akumulátoru

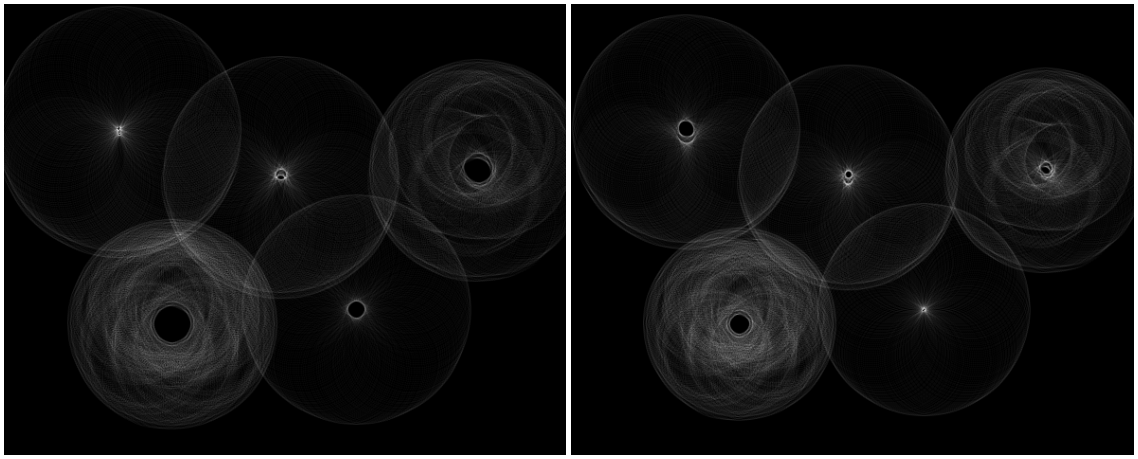
7. Následně dojde k prohledání naplněného akumulátoru. Pro všechny body, jejichž hodnota je vyšší než nastavený práh, bude do výsledného obrazu vykreslen fialový bod a červená kružnice o poloměru r, které značí střed a obvod kružnice.

```
Scalar red(0, 0, 255), purple(255, 0, 255);
for(int i = 0; i < Accumulator.rows; i++) {
    for(int j = 0; j < Accumulator.cols; j++) {
        if(Accumulator.at<uchar>(i, j) >= Treshold) {
            circle(*Result, Point(j,i), 1, purple, 2, LINE_AA);
            circle(*Result, Point(j,i), r, red, 2, LINE_AA);
        }
    }
}
AccumulatorVector->push_back(new Mat(Accumulator));
```

Výpis 4.7: Prohledání akumulátoru a vykreslení kružnic do obrazu

Po prohledání se akumulátor uloží do připraveného pole a výpočet se opakuje pro další testovaný poloměr.

8. Pro ukázku tu jsou připraveny obrazy akumulátorů pro dva různé poloměry. (obr. 4.7) Na obrázcích je dobře vidět, jak se vykreslované kružnice protaly v jednom bodě a odhalily tak střed původních hledaných kružnic.



Obr. 4.7: Detekce kružnic – ukázka akumulátoru pro různé poloměry kružnic

9. Výsledná fotografie s detekovanými kružnicemi je zobrazena na obrázku 4.8, práh byl pro tento výsledek nastaven na hodnotu 140.

4.3 Implementace detekce obecných těles

1. Pro ukázku jsem si vybral tuto fotografii náměstí Nového Města na Moravě (obr. 4.9). Tato fotografie pochází z [13].
2. V obraze se nachází značné množství objektů, které by bylo možné detekovat. Vyberu například toto zaparkované auto značky Škoda (obr. 4.10).

Obr. 4.8: Detekce kružnic – výsledná fotografie s nalezenými kružnicemi

Obr. 4.9: Detekce obecných těles – zvolená fotografie

Obr. 4.10: Detekce obecných těles – hledaný objekt

3. Pro detekci algoritmus vyžaduje binární obrazy detekovaných hran jak fotografie, tak hledaného objektu. Bude postupováno podle návodu zmíněném v sekci 3. To znamená, že obraz bude převeden na stupně šedi, aplikuje se Gaussův filtr

a následně se použije Cannyho detektor hran. V kódu jsem tyto operace implementoval jako lambda funkci, jelikož je potřebuji uplatnit vícekrát, ale pouze v rámci této metody.

```
auto DetectEdges = [](Mat& input) {
    Mat mat;
    cvtColor(input, mat, COLOR_BGR2GRAY);
    GaussianBlur(mat, mat, Size(5,5), 0);
    Canny(mat, mat, 100, 150, 3);
    return new Mat(mat);
};
if(!Picture->empty())
    FoundEdges = DetectEdges(*Picture);
if(!Template->empty())
    FoundEdgesTemplate = DetectEdges(*Template);
}
```

Výpis 4.8: Detekce hran ve fotografii a objektu

Výsledné binární obrazy detekovaných hran se nachází na obrázcích 4.11 a 4.12.

Obr. 4.11: Detekce obecných těles – nalezené hrany ve fotografii

Obr. 4.12: Detekce obecných těles – nalezené hrany hledaného objektu

4. Vytvořím si pole derivací podle osy x a y. Pole P_x, P_y budou obsahovat hodnoty derivace pixelů originální fotografie a pole O_x, O_y hodnoty derivace pixelů hledaného objektu. Ty budu potřebovat pro výpočet gradientu nebo-li sklonu hrany. K jejich vypočtení využiji funkci *Sobel()* obsaženou v knihovně *OpenCV*. Aproximace této funkce je sice poměrně hrubá, ale pro mé využití více než dostačující. Pro její využití je třeba obraz nejdříve převést na stupně šedi a následně aplikovat Gaussův filtr pro vyhlazení. Jelikož tuto operaci musím dohromady provést čtyřikrát a nikde jinde v kódu ji neuplatním, tak jsem i v tomto případě, stejně jako u detekce hran, využil možnosti vytvoření lambda funkce. Ta existuje pouze v rámci její nadřazené funkce, metody.

```
auto calculateGradient = [](Mat& input, int dx, int dy){
    Mat mat;
    Mat* output = new Mat();
    cvtColor(input, mat, COLOR_BGR2GRAY);
    GaussianBlur(mat, mat, Size(3,3), 0);
    Sobel(mat, *output, CV_16S, dx, dy, 3);
    return output;
};
if(!Picture->empty()) {
    GradPictureX = calculateGradient(*Picture, 1, 0);
    GradPictureY = calculateGradient(*Picture, 0, 1);
}
if(!Template->empty()) {
    GradTemplateX = calculateGradient(*Template, 1, 0);
    GradTemplateY = calculateGradient(*Template, 0, 1);
}
```

Výpis 4.9: Výpočet gradientu

5. Pro základní verzi zobecněné detekce těles bude dostačující dvourozměrný akumulátor $A(x, y)$. Pokud bych však chtěl vzít v potaz i natočení nebo změnu měřítka hledaného objektu, bylo by třeba zvýšit rozměry pole o jednu dimenzi pro každou z operací. Ale proto, že implementace Ballardovy zobecněné Houghovy transformace, která se nachází v knihovně OpenCV, tyto operace v potaz nebrala, rozhodl jsem se také implementovat pouze základní verzi. Vytvořím si tedy dvourozměrný akumulátor, jehož rozměry budou dány rozměry vstupní fotografie.
6. Dále je potřeba připravit R-table (LuT – Look up Table), který byl zmíněný v teoretické části detekce obecných těles 2.1.1. Je tvořen dvourozměrným polem o rozměrech $(180 \times n)$, kde n je celkový počet bodů, které mají stejný sklon úhlu k pomyslnému středu a 180 je počet úhlů. Důvod, proč nevytvářet R-table pro všech 360 °, je ten, že pro popsání sklonu přímky nám stačí pouze dva sousední kvadranty kartézského souřadného systému, tedy 180 °.
7. V binárním obraze objektu je třeba zvolit referenční bod. Jeho souřadnice mohou

být libovolné, avšak nejčastěji se volí střed obrazu, tedy bod o souřadnicích

$$\left(\frac{\text{šířka}}{2}, \frac{\text{výška}}{2}\right)$$

a ten si zvolím pro tuto implementaci i já.

8. Algoritmus začne s plněním LuT. Existují dva základní způsoby, jak hodnoty vzdálenosti ukládat. Je možné ukládat vzdálenost r vypočítanou pomocí Pythagorovy věty a úhel α, který r svírá s poloosou x nebo ukládat délku x-ové a y-ové složky vzdálenosti. Pro zjednodušení a snížení množství přepočtů je implementována druhá možnost, ukládání rozdílu souřadnic středu a hrany vypočítaných podle

$$\Delta x = x_s - x_h,$$
$$\Delta y = y_s - y_h,$$

kde $\Delta x, \Delta y$ jsou vzdálenosti od středu, x_s, y_s jsou souřadnice středu a x_h, y_h jsou souřadnice hrany.
Algoritmus prochází binární obraz hledaného objektu, při nalezení hrany vypočítá její gradient ϕ podle

$$\phi = \arctan\left(\frac{O_y(x,y)}{O_x(x,y)}\right)$$

a dále hodnoty $\Delta x, \Delta y$. Ty uloží do LuT jako funkci gradientu $LuT(\phi)$. Postup opakuje pro všechny hrany v obraze.

```
vector<vector<Point>>LUT(180,vector<Point>(0));
for(int i = 0; i < FoundEdgesTemplate->rows; i++) {
    for(int j = 0; j < FoundEdgesTemplate->cols; j++) {
        if( FoundEdgesTemplate->at<uchar>(i,j) == 255 &&
            GradTemplateX->at<short>(i,j) != 0 ) {
            int phi = 90 + CONVDEG * atan (
                    GradTemplateY->at<short>(i,j) /
                    (double)GradTemplateX->at<short>(i,j)
                    );
            LUT[phi].push_back(Point(center.x-j, center.y-i));
        }
    }
}
```

Výpis 4.10: Naplnění LuT

9. LuT byl naplněn a nyní je třeba najít hledaný objekt ve vstupním obrázku. Algoritmus prochází binární obraz a pro každou nalezenou hranu vypočítá ϕ podle

$$\phi = \arctan\left(\frac{P_y(x,y)}{P_x(x,y)}\right).$$

Následně pro všechny body $b = LuT(\phi)$ inkrementuje akumulátor na pozici $A(x + b_x, y + b_y)$. Tyto kroky opakuje pro všechny hrany v obraze.

```
for(int i = 0; i < FoundEdges->rows; i++) {
    for(int j = 0; j < FoundEdges->cols; j++) {
        if( FoundEdges->at<uchar>(i,j) == 255 &&
            GradPictureX->at<short>(i,j) != 0 ) {
            int phi = 90 + CONVDEG * atan(
                      GradPictureY->at<short>(i,j) /
                      (float)GradPictureX->at<short>(i,j));
            for(Point Edge: LUT[phi]) {
                if(notOutOfBounds(Edge.x + j, Edge.y + i))
                    Accumulator->at<int>(Edge.y+i,Edge.x+j)++;
            }
        }
    }
}
```

Výpis 4.11: Naplnění akumulátoru

Naplněný akumulátor je možné vidět na obrázku 4.13. V levé spodní části se nachází světlý „bod“ přesně tam, kde by se měl nalézat zvolený střed objektu. Důvod, proč všechny uložené vzdálenosti neinkrementovali pouze referenční bod,

Obr. 4.13: Detekce obecných těles – naplněný akumulátor

ale i buňky okolo je ten, že hodnot se stejným ϕ se v LuT obvykle nachází víc než jedna. A jelikož inkrementujeme pro všechny uložené hodnoty, bude docházet k odchylkám. Na obrázek byla aplikována logaritmická jasová transformace pro zvýraznění.

10. V posledním kroku algoritmus nalezne v akumulátoru body, které mají vyšší hodnotu než je zvolený práh a ty bude považovat za středy hledaného objektu. Kolem každého nalezeného středu vykreslí červený obdélník a tím ve fotografii vyznačí hledaný objekt.

```
Scalar red = Scalar(0,0,255);
for(int i = 0; i < Accumulator->rows; i++) {
    for(int j = 0; j < Accumulator->cols; j++) {
        if(Accumulator->at<int>(i,j) > Treshold) {
            Point p1(j + Template->cols/2, i + Template->rows/2),
                  p2(j - Template->cols/2, i - Template->rows/2);
            rectangle(*Result, p1, p2, red, 1, LINE_AA);
        }
    }
}
```

Výpis 4.12: Prohledání akumulátoru

Obr. 4.14: Detekce obecných těles – výsledný obraz s nalezeným objektem

5 Výsledky studentské práce

5.1 Program

Hlavním výsledkem studentské práce je přiložený program, který dokáže, po načtení fotografie a specifikování parametrů, detekovat přímky, kružnice nebo hledaný objekt, případně hledané objekty v obraze.

Projekt obsahuje tyto třídy a soubory:

1. main – hlavní soubor sloužící k inicializaci uživatelského rozhraní.
2. HoughTransform – hlavní třída obsahující všechny grafické prvky uživatelského rozhraní. Mimo jiné také obstarává načítání ukázek z resource souboru a zobrazování výsledků po detekci.
3. MyHough – abstraktní třída obsahující ukazatele na načtený obrázek, obraz detekovaných hran, akumulátoru a výsledku. Také obsahuje virtuální metody využívané všemi detekcemi.
4. MyHoughLines – dědí z abstraktní třídy MyHough, obsahuje metodu s implementovaným algoritmem detekce přímek a dále metody pro přípravu potřebných hodnot pro detekci.
5. MyHoughCircles – přebírá metody z abstraktní třídy MyHough, přetěžuje některé z těchto metod a definuje nové metody pro přípravu hodnot pro detekci. Obsahuje také metodu s implementovaným algoritmem detekce kružnic.
6. MyGeneralizedHoughBallard – také dědí z abstraktní třídy MyHough a přetěžuje některé z virtuálních metod, přidává ukazatele na obrázek hledaného objektu. Dále také obsahuje metodu detekující sklon hran a metodu implementující algoritmus detekce obecných těles.

5.2 Časová náročnost výpočtu

Výsledný čas byl vypočítán jako průměr hodnot získaných ze sta opakovaných měření. Tyto hodnoty byly naměřeny na notebooku s šestijádrovým procesorem Intel i7 8750h a Samsung 2×8 GB DDR4 2667 MHz ram.

5.2.1 Detekce přímek

Po provedení optimalizací zmíněných v sekci 4.1, se mi podařilo snížit výpočetní náročnost detekce přímek na stejnou úroveň, jakou poskytuje implementace dostupná v knihovně OpenCV.

Rozlišení	640 x 360	1280 x 720	1920 x 1080
-	čas [ms]		
OpenCV verze	5,78	14,08	22,71
Moje verze	5,53	14,03	22,25

Tab. 5.1: Časová náročnost detekce přímek

5.2.2 Detekce kružnic

Rozlišení	640 x 360	1280 x 720	1920 x 1080
-	čas [ms]		
OpenCV verze	5,42	19,56	37,77
Moje verze : 30 bodů	17,42	53,33	103,74
Moje verze : 90 bodů	32,28	92,57	165,52
Moje verze : 150 bodů	49,28	135,02	234,95

Tab. 5.2: Časová náročnost detekce kružnic

Při měření časové náročnosti bylo testováno, při každém ze sta průchodů, 25 poloměrů. Z tabulky naměřených hodnot je vidět, že i v nejlepším případě je moje implementace 3x pomalejší než implementace v knihovně OpenCV. Je tomu ze dvou hlavních důvodů.

Prvním důvodem je, že implementace v knihovně OpenCV využívá metodu Houghova gradientu, která sama o sobě je značně rychlejší pro velké množství testovaných poloměrů. Každopádně, tato její výhoda se nijak výrazně neuplatnila, jelikož jsem ve fotografii testoval pouze 25 různých poloměrů. A druhým, značně podstatnějším, důvodem je, že implementace v knihovně OpenCV využívá paralelního zpracování dat. Využití paralelizace umožňuje zrychlit výpočet přibližně 6,8x na čtyřjádrovém procesoru.

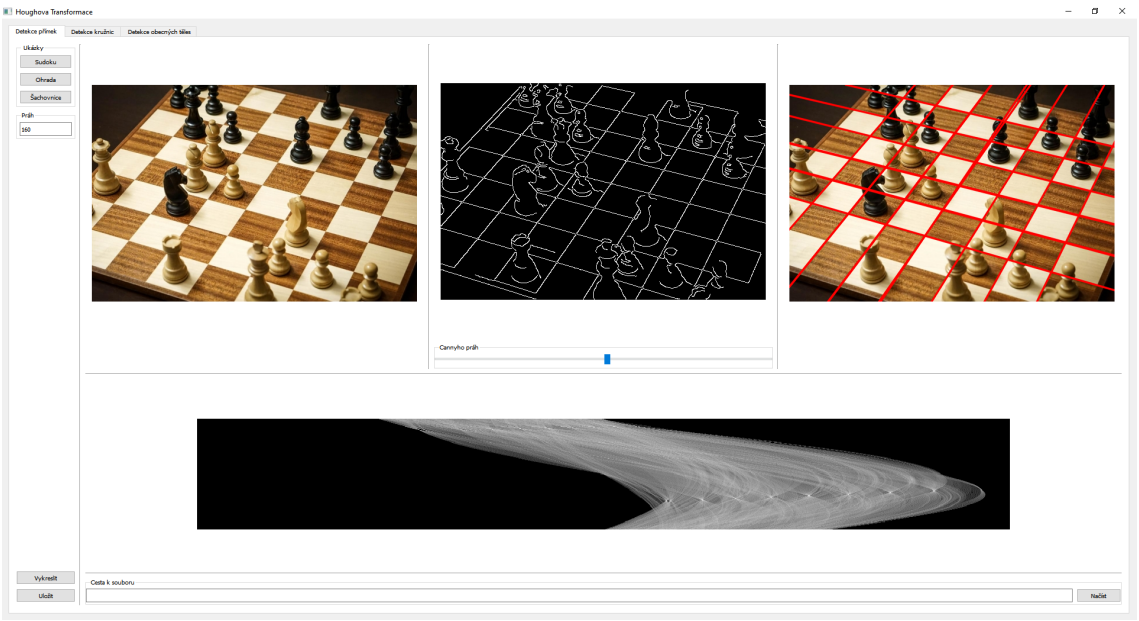
5.2.3 Detekce obecných těles

Podle naměřených hodnot by měla být moje implementace detekce obecných těles přibližně 2x rychlejší než detekce implementovaná v knihovně OpenCV. To je však způsobeno tím, že OpenCV verze Houghovy transformace provádí i detekci hran v rámci dané metody. V momentě, kdy jsem změřil výpočetní čas své implementace společně s algoritmem detekce hran, se naměřené hodnoty lišily minimálně. Jediná odchylka naměřených hodnot se objevila při testování fotografie s HD rozlišením, kdy moje implementace byla průměrně o 4,5 ms rychlejší.

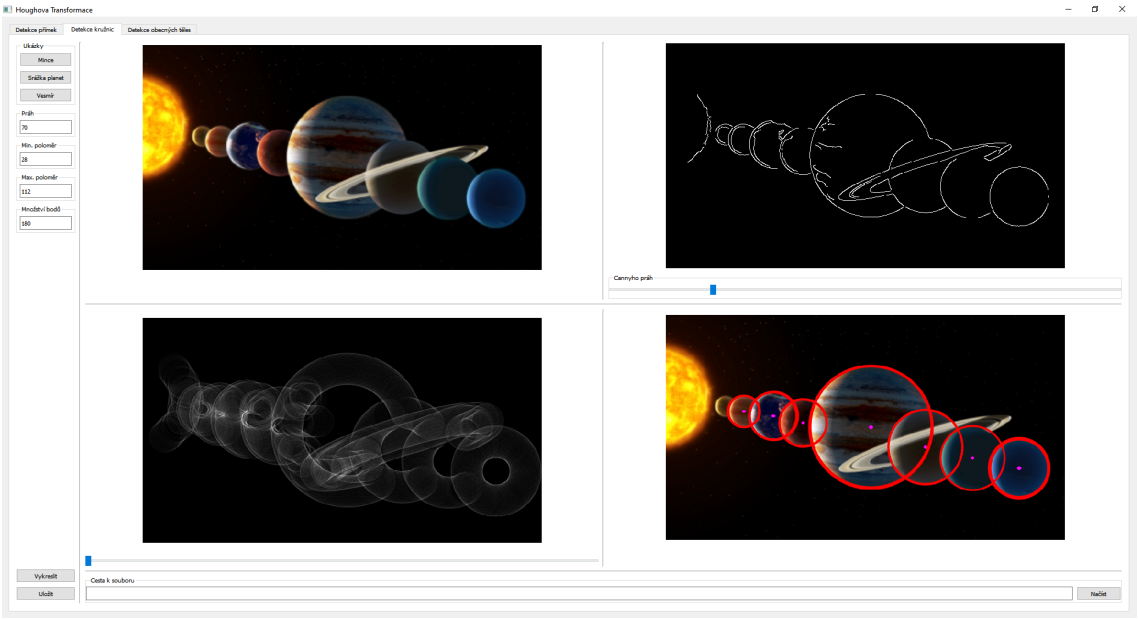
Rozlišení	640 x 360	1280 x 720	1920 x 1080
-	čas [ms]		
OpenCV verze	5,92	17,77	33,57
Moje verze	2,63	8,77	19,36
Moje verze + detekce hran	5,49	13,15	34,29

Tab. 5.3: Časová náročnost detekce obecných těles

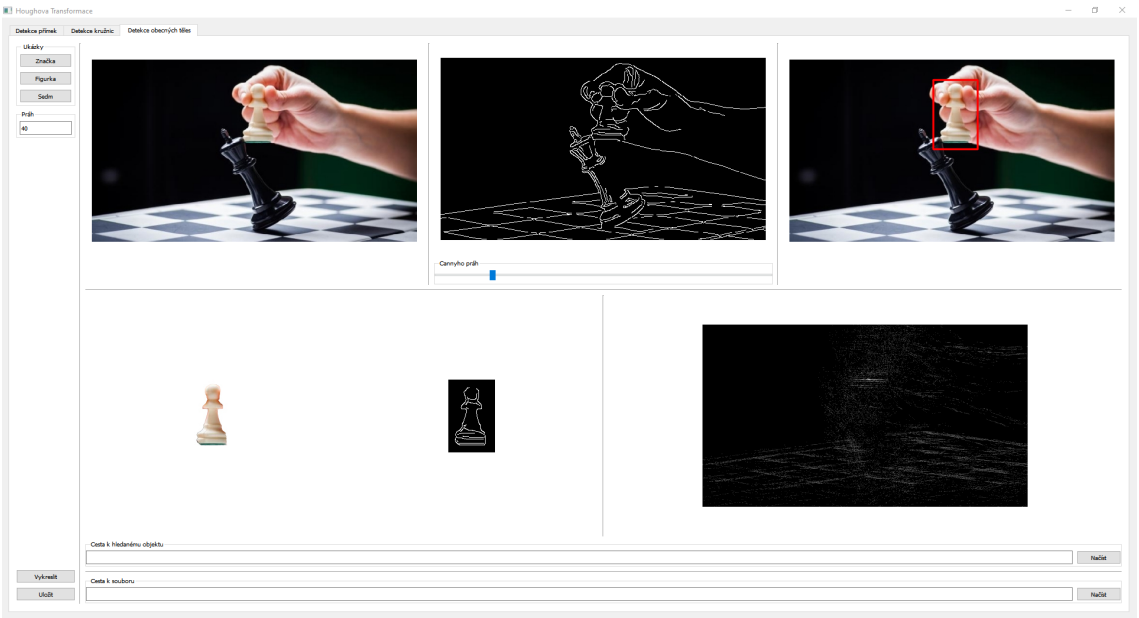
5.3 Obrázky uživatelského rozhraní aplikace



Obr. 5.1: Uživatelské rozhraní – detekce přímek



Obr. 5.2: Uživatelské rozhraní – detekce kružnic



Obr. 5.3: Uživatelské rozhraní – detekce obecných těles

Závěr

Nastudoval jsem teoretické principy Houghovy transformace a její různé varianty. Přesněji se jednalo o detekci přímek, kružnic a obecných těles v obraze. Dále jsem vytvořil aplikaci, která implementuje dané algoritmy. Umožňuje načíst vlastní obrázek, provádět detekci jak v připravených ukázkových obrázcích, tak ve vlastním načteném obrázku, měnit parametry detekce a výsledné obrázky uložit.

K implementaci algoritmů jsem využil doporučenou knihovnu OpenCV a programovací jazyk MS Visual C++. Grafické rozhraní aplikace jsem vytvořil za pomoci IDE Qt Creator, který nabízí snadnou „drag and drop“ metodu tvorby uživatelského rozhraní.

Výsledky implementovaných algoritmů detekce přímek a obecných těles jsou uspokojující. Z hlediska časové náročnosti výpočtu se vyrovnají implementacím nabízeným knihovnou OpenCV a co se přesnosti detekce týče, tak nezaostávají pozadu a dokážou dosáhnout velice podobného výsledku.

Algoritmus detekce kružnic je mírným zklamáním, jelikož mnou implementovaný kód je značně pomalejší než její dostupné alternativy. Jak jsem již zmínil, je tomu tak hlavně proto, že implementace v knihovně OpenCV využívá paralelního výpočtu. Z hlediska přesnosti detekce však obě implementace dosahují velice podobného výsledku.

V rámci diplomové práce by bylo možné tuto práci rozšířit o implementaci jiných metod detekce přímek, kružnic a obecných těles v obraze, jako je například pravděpodobnostní Houghova transformace. Jejich následné testování, porovnání a shrnutí získaných výsledků. Dalším z možných způsobů, jak tuto práci rozšířit, by bylo porovnat výsledky získané těmito běžnými metodami a výsledky získané využitím moderních neuronových sítí. Následně na základě výsledků posoudit, zda-li se Houghovu transformaci v dnešní době stále vyplatí používat.

Literatura

[1] DUDA, Richard O. a Peter E. HART. *Use of the Hough Transformation To Detect Lines and Curves in Pictures* [online]. Stanford Research Institute, Menlo Park, California, 1972 [cit. 2020-11-28]. Dostupné z: https://www.cse.unr.edu/~bebis/CS474/Handouts/HoughTransformPaper.pdf

[2] PEDERSEN, Simon Just Kjeldgaard. *Circular Hough Transform* [online]. Aalborg University, Vision, Graphics, and Interactive Systems, 2007 [cit. 2020-11-29]. Dostupné z: https://cdn.manesht.ir/9961___Simon_Pedersen_CircularHoughTransform.pdf

[3] KANG & ATUL. *Hough Circle Transform.* TheAILearner [online]. 24.11.2020 [cit. 2021-5-14]. Dostupné z: https://theailearner.com/tag/hough-gradient-method/

[4] Balard, D.H. *GENERALIZING THE HOUGH TRANSFORM TO DETECT ARBITRARY SHAPES* [online]. Computer Science Department, University of Rochester, Rochester, NY 14627, U .S .A, 1980 [cit. 2020-11-27]. Dostupné z: http://www.eng.tau.ac.il/~cvapps/Supplement/%5B%201981%20%5D%20Generalizing%20the%20Hough%20Transform%20to%20Detect%20Arbitrary%20Shapes.pdf

[5] *Generalised Hough transform.* Wikivisually.com [online]. [cit. 2020-11-11]. Dostupné z: https://wikivisually.com/wiki/Generalised_Hough_transform

[6] *Article 9: Gaussian filter, or Gaussian blur.* Librow [online]. 2007- [cit. 2020-11-29]. Dostupné z: http://www.librow.com/articles/article-9

[7] *Gaussian blur.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-11-29]. Dostupné z: https://en.wikipedia.org/wiki/Gaussian_blur

[8] SOBEL, Irwin. *An Isotropic 3x3 Image Gradient Operator.* ReserchGate [online]. Únor 2014 [cit. 2020-11-25]. Dostupné z: https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator

[9] *Sobel Operator.* Tutorialspoint: simplyeasylearning [online]. [cit. 2020-11-30]. Dostupné z: https://www.tutorialspoint.com/dip/sobel_operator.htm

[10] *Sobel operator.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-11-25]. Dostupné z: https://en.wikipedia.org/wiki/Sobel_operator

[11] *Canny edge detector.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2020 [cit. 2020-11-30]. Dostupné z: https://en.wikipedia.org/wiki/Canny_edge_detector

[12] *Grayscale.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2020 [cit. 2020-11-29]. Dostupné z: https://en.wikipedia.org/wiki/Grayscale

[13] *File:Náměstí (Nové Město na Moravě).JPG.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-11-25]. Dostupné z: https://commons.wikimedia.org/wiki/File:N%C3%A1m%C4%9Bst%C3%AD_(Nov%C3%A9_M%C4%9Bsto_na_Morav%C4%9B).JPG

A Obsah přiloženého souboru

V přiloženém souboru můžete nalézt:

- `/` ..... kořenový adresář
  - `Houghova Transformace` ..... projektová složka programu
    - `HoughovaTransformace.pro` ..... projektový soubor
    - `HoughovaTransformace.pro.user`
    - `main.cpp`
    - `houghtransform.h`
    - `houghtransform.cpp`
    - `houghtransform.ui`
    - `myhough.h`
    - `myhough.cpp`
    - `myhoughlines.h`
    - `myhoughlines.cpp`
    - `myhoughcircles.h`
    - `myhoughcircles.cpp`
    - `mygeneralizedhoughballard.h`
    - `mygeneralizedhoughballard.cpp`
    - `Obrazky.qrc`
    - `Obrazky` ..... ukázkové obrázky pro program
      - `coins.jpg`
      - `chess1.jpg`
      - `chessFigure1.jpeg`
      - `chessFigureTemplate1-1.jpg`
      - `line_det1.jpg`
      - `planets_coliding1.jpg`
      - `space1.jpg`
      - `Stop1.jpg`
      - `StopTemplate1.jpg`
      - `sudoku400.jpg`
      - `sudoku1024-2.jpg`
      - `sudokuSeven-400.jpg`

Kód byl testován v Qt Creator 4.13.2 (Community).